Learning 2-additive Hierarchical Choquet Integrals with non-monotonic utilities

Roman Bresson\textsuperscript{1}, Johanne Cohen\textsuperscript{2}, Eyke Hüllermeier\textsuperscript{3}, Christophe Labreuche\textsuperscript{4}, Michèle Sebag\textsuperscript{5}

Abstract. The Choquet integral, a main ingredient in Multi-Criteria Decision Aid systems, is meant to capture interactions among criteria while enforcing the interpretability of the full aggregated model. The scalability of the approach when the number of criteria increases is achieved using the so-called Hierarchical Choquet integrals (HCI), gradually aggregating the interacting criteria along some predefined tree-structure. This paper extends the \textsc{Neur}-\textsc{Hci} framework first presented in [2], concerned with the automatic identification of hierarchical MCDA models and underlying marginal utilities from data. \textsc{Neur}-\textsc{Hci} was restricted to 2-additive Choquet integral aggregators and monotonic marginal utilities, assuming that the criteria tree-structure to be known. The contribution of this paper are the ability for \textsc{Neur}-\textsc{Hci} to learn new types of utilities, along with being able to automatically select the best-fitting type.

1 Introduction

Among the wide families and applications of predicting models, Decision Aid models are particular, as their output will not be used by themselves, but will serve as the basis for an expert Decision Makers (DM) to make a choice in a complex situation. It is thus important, in many cases, that the models be interpretable, so that the DM can understand why an output has a certain value. Moreover, for the DM to trust a model, it might be necessary it fulfills some formal constraints (i.e. monotonicity), most often coming from some field knowledge.

Multicriteria Decision Aid makes significant use of the Choquet Integral (CI), an aggregation function which generalizes the weighted sum by allowing to represent interactions between the aggregated criteria. CIs are often used in order to compute the global score of an alternative from its local scores, computed independently on each criterion.

We focus in this paper on 2-additive Choquet Integrals, a subset of the CIs which only considers interactions between up to two criteria at a time. This allows to retain a higher level of interpretability, along with keeping a low number of parameters (quadratic in 2-additive CIs, exponential in the general case).

Despite this restriction, large aggregations, with numerous criteria, can become quite difficult to interpret for a decision maker (DM). In order to allow the DM to easily comprehend why an output has a given value. One way of coping with this issue is to use hierarchical models. These models use a divide-and-conquer approach in order to split the aggregation into several sub-aggregations, following a tree-like structure. Each such sub-aggregation computes the score of the alternative on a subset of criteria. All of these are then aggregated, again, until a single, global score remain. From now on, we call 2-additive Hierarchical Choquet integrals (HCIs) such hierarchical models where all of the aggregations are 2-additive CIs.

Such models can be enriched through the use of criteria-wise rescalings, also known as marginal utilities. Marginal utilities take as inputs the raw values of the criteria, and return a vector of criteria-wise satisfactions, which will then serve as an input for an aggregation function. Marginal utilities are usually heavily constrained, either monotonic, or bitonic (single-peaked/single valleyed). These constraints allow two things: firstly, they often come from domain knowledge, and it is thus necessary for a DM that they be respected if the model is to be trusted; secondly, they ensure a simple behaviour for the marginal utilities, and allow interpretability. HCIs fitted with marginal utilities are called, below, UHCIs.

Artificial neural networks (ANNs) are powerful tools for learning the parameters of a model from data. Leveraging the power of backpropagation, they are particularly adapted when inferring the parameters of a hierarchical model. Indeed, such models are compositions of diverse functions, as is, in essence, an ANN. Nonetheless, ANNs are often heavily over-parameterized, which prevents a straightforward interpretation. Moreover, their complex structure, where all parameters interact with each other, prevents a decision maker from obtaining a “local” interpretation of a phenomenon, as each parameter is meaningless on its own.

Another difficulty arises as UHCIs are heavily constrained models. Such formal constraints (i.e. monotonicity, ordering of parameters) must be enforced through the whole model, preventing a “free” gradient descent, and leading to some instability if enforced through hard measures, such as clipping, or projecting the model in a “constraint-fitting” state every time it violates one of them.

A way to learn Hierarchical 2-additive Choquet Integrals through ANNs has been proposed by [2]. This approach is able to learn the parameters of such hierarchical models, along with marginal utility functions. The main restriction on the latter is that the learned marginal utilities were necessarily monotonic; moreover, the monotonicity had to be given beforehand by an expert advisor.

We propose here an extension of this work, which consists in new marginal utility neural networks, or modules. The improvements are twofold:

- the marginal utilities can now be single-peaked (resp. single-valleyed), meaning non-decreasing, then non-increasing (resp non-increasing then non-decreasing). We use the term "bitonic" to designate both configurations
- the type of marginal utility (among non-decreasing, non-increasing, single-peaked, single-valleyed), needs not be given by an expert and fixed before training, it can now be learned along with the

\textsuperscript{1} Thales Research and Technology, email:romain.bresson@thalesgroup.com
\textsuperscript{2} Laboratoire de Recherche en Informatique, email:johanne.cohen@lri.fr
\textsuperscript{3} Paderborn University, email:eyke@upb.de
\textsuperscript{4} Thales Research and Technology, email:christophe.labreuche@thalesgroup.com
\textsuperscript{5} Laboratoire de Recherche en Informatique, email:michele.sebag@lri.fr
parameterization

Example 1: In a medical context, a physician must evaluate a patient’s health on a scale from 0 to 10 (the higher, the healthier) based on various criteria. They have some strong a priori constraints as to how the health grade must change w.r.t. some of the criteria. For instance, it must:

- increase w.r.t. the oxygen saturation in the blood
- decrease w.r.t. the cholesterol levels
- behave as a peak w.r.t. the heart rate (a heart that beats too slow is dangerous (bradycardia), so is one that beats too fast (tachycardia), while an average rate indicates good heart functions).

A UHCl could represent such preferences, using a non-decreasing (resp. non-increasing, resp. marginal utility) for criterion oxygen saturation (resp. cholesterol, resp. heart rate).

Such extensions offer some generalization capabilities, and mostly a lot of additional autonomy to the Neur-HCI framework. Moreover, as for the monotonic marginal utilities, these new modules represent functions which meet all of the constraints expected from such utility functions (described in section 2).

The paper is organized as follows. The formal context and relevant notes on the Neur-HCI framework can be found in section 2. Then the state of the art of current methods will be described in 3. Section 4 describes our improvements to Neur-HCI, and section 5 will provide empirical results of our improvements, on artificial benchmarks.

2 Background

We present in this section the formal setting in which this paper stands, along with a quick description of the Neur-HCI framework.

2.1 Notations

Let $N = \{1, \ldots, n\}$ denote a set of criteria, with $X_i$ the domain of the $i$-th criterion.

Example 2: A company wishes to add a car to its fleet. The manager must select a car model based on 4 criteria: (1) autonomy (in kilometers), (2) price (in dollars), (3) size (in meters), (4) fuel type (either electrical, hybrid, or diesel).

While it is obvious that $X_i \subseteq \mathbb{R}$ for $i \in 1, 2, 3$, one can build $X_4 = \{0, 1, 2\}$ through a mapping $m$ such that $m(\text{electrical}) = 0$, $m(\text{hybrid}) = 1$, $m(\text{diesel}) = 2$. An expert might be needed to order such values so that the preference to increase (or decrease) with it.

Alternatives are defined as elements in $X = X_1 \times \cdots \times X_n$. We call a decision model a function which can do either of these tasks:

- select the preferred alternative from a set;
- order alternatives from most preferred to least preferred;
- map alternatives to a given preference class;

where the preference relation corresponds to that of a human decision maker.

A scoring model $S : X \to Y$ where $Y = [0, 1]$ maps a score $S(x)$ to an alternative $x$. It makes it possible to perform the three tasks above. Indeed, we now define a complete preference order $\succeq$ among alternatives:

$$a \succeq b \iff S(a) > S(b)$$

with $(a, b) \in X^2$. Here, we focus on so-called decomposable scoring models [8], where $S$ can be written as:

$$S(x) = A(u_1(x_1), \ldots, u_n(x_n))$$

where $A : [0, 1]^n \to Y$ is called an aggregation function, and for each criterion $i$ in $N$, $u_i : X_i \to [0, 1]$ is said to be a marginal utility function.

2.2 Marginal utility functions

We denote by $\geq_i$ the preference relation with respect to the $i$-th criterion. This means that, if two alternatives $a$ and $b$ have the same value on all criteria except $i$, where they have respectively values $a_i$ and $b_i$, then:

$$a_i \geq_i b_i \implies a \geq b$$

Marginal utilities, or marginal utility functions, aim at representing such marginal preferences. Let $i \in N$, let $(a_i, b_i) \in X_i^2$, then

$$u_i(a_i) \geq u_i(b_i) \iff a_i \geq_i b_i$$

We consider here two usual types of marginal utilities:

- monotonic: non-decreasing or non-increasing
- bitonic: single-peaked (non-decreasing, then non-increasing), or single-valleyed (non-increasing, then non-decreasing)

Example 3: In Ex. 2, the utilities that represent the preferences of the DM are: (1) non-decreasing (the more autonomy, the better); (2) non-increasing (the cheaper, the better); (3) single-peaked (the car should neither be too small to be comfortable, nor too big to easily park and maneuver); (4) non-decreasing (electric is preferred).

Let $i$ be a criterion. We add the following normalization constraints on such a function $u_i$: there exists two elements $x_i, x'_i$ in $X_i$ such that $u_i(x_i) = 0$ and $u_i(x'_i) = 1$. Moreover, the value at the extremities (which might be infinite) of $X_i$ are either 0 or 1.

The motivation for keeping such simple marginal utilities is twofold. Firstly, it makes interpretation easier while allowing to represent an important number of real-life cases. Secondly it provides natural regularization for the problems ahead, avoiding some of the over-fitting that would occur if more complex behaviors were allowed.

2.3 Aggregation functions

2.3.1 2-additive Choquet Integral

Due to Eq. (1) and (2), we want our aggregation function $A$ to be non-decreasing w.r.t. its inputs : given $(u, v) \in [0, 1]^n$, such that $\forall i \in \{1, \ldots, n\}$, $u_i \geq v_i$, then $A(u) \geq A(v)$.

It fits with the idea that the marginal utilities represent the criteria-wise satisfaction; as a consequence, an increase in a local satisfaction can only lead to an increase in the global satisfaction.

In our case, we use 2-additive Choquet Integrals (CI) [7] as aggregation functions. A 2-additive CI, parameterized by a set of weights $w$, on an alternative $a$ defined on $n$ criteria can be written as:

$$C_w(a) = \sum_{i=1}^{n} w_i a_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} (w_i \wedge a_i \wedge a_j) + w_i \lor (a_i \lor a_j)$$

where $\wedge$ (resp $\lor$) denote the min (resp max) operators and where we have:

$$\wedge(a, b) = \min(a, b)$$

$$\lor(a, b) = \max(a, b)$$

with $(a, b) \in X^2$. Here, we focus on so-called decomposable scoring models [8], where $S$ can be written as:

$$S(x) = A(u_1(x_1), \ldots, u_n(x_n))$$

where $A : [0, 1]^n \to Y$ is called an aggregation function, and for each criterion $i$ in $N$, $u_i : X_i \to [0, 1]$ is said to be a marginal utility function.
• Monotonicity: \( \forall i \in N, \forall j \in N, \forall k \in N, w_{i,k} \geq 0, w_{i,j} \geq 0, w_{i,j} \geq 0 \)

• Normalization: \( \sum_{i=1}^{n} w_{i} + \sum_{j=1}^{n} \sum_{j=1}^{n} (w_{i,j} + w_{i,j}) = 1 \)

2.3.2 Hierarchical 2-additive Choquet Integrals

The Choquet Integral is a compensatory operator, which means that it has the following interesting property:

\[
\min(a_i) \leq C_N(a) \leq \max(a_i)
\]

(4)

where \( a \) is an alternative and \( a_i \) is its \( i \)-th attribute.

This allows us to build a hierarchical model by aggregating sequentially, in a tree-like fashion, several disjoint subsets of criteria by 2-additive CIs, then aggregating these outputs, and so on until a single score remains. We call this a 2-additive Hierarchical Choquet Integral (HCl).

Example 3: Assume the cars from Ex. 2 are now evaluated on 6 criteria, with (1) to (4) remaining unchanged, and adding (5) the maintenance cost, and (6) the number of seats.

Then the criteria can be hierarchized as follows: Criteria (1) and (4) can be aggregated into artificial criterion (7) fuel consumption, (2) and (5) into artificial criterion (8) total cost, criteria (3) and (6) into artificial criterion (9) comfort, then (7), (8) and (9) are aggregated into the global score (see Figure 1).

![Figure 1. Hierarchy of criteria for Example described in Section 2.1.](image)

It is easy to see that such a hierarchical operator conserves the non-decreasingness w.r.t. its inputs, along with the compensatory property. Note that the number of levels and of children (degree) for each node can be arbitrary. Nonetheless, it should be kept in mind that a low degree (less than around 7) makes an aggregator much easier to understand for a DM.

2.4 Neur-HCI

Neur-HCI is a framework, developed in [2], which allows to learn the parameters (weights and monotonic marginal utilities) of a 2-additive HCIs, given data, a hierarchy, and the monotonicity of each marginal utility. It does so by leveraging the regression power of artificial neural networks (ANNs), while enforcing all of the constraints required above.

We describe briefly below the two types of modules implemented by Neur-HCI: the 2-additive CI module, and the monotonic utility module.

2.4.1 2-additive Choquet Integral module

A 2-additive CI module is a small multilayer perceptron which implements a 2-additive CI, in the shape given in Eq. (3). Assume a module with \( n \) inputs. The hidden layer will be \( n^2 \) neurons wide (see Figure 2), divided into three categories:

- category A: \( n \) neurons receiving each a single input and returning it unchanged;
- category B: \( \frac{n(n-1)}{2} \) neurons, receiving each a different pair of inputs \( u_i, u_j \), and returning \( u_i \land u_j \);
- category C: \( \frac{n(n-1)}{2} \) neurons, receiving each a different pair of inputs \( u_i, u_j \), and returning \( u_i \lor u_j \).

This is analogous to a pooling layer. Then, the outputs of the hidden nodes A, B and C are respectively weighted by the weights \( w_{i,j}, w_{i,j}^1, w_{i,j}^2 \) and \( w_{i,j}^3 \).

The constraints on the weights (see Section 2.3.1) are met by construction. Non-negativity is ensured by building the weights as the image by a non-negative mapping (softplus) of the real learned parameters. Normalization is ensured through batch-renormalization.

Note that in the case of a hierarchical model, the output node of a Choquet module will be one of the input nodes of the module representing its parent in the hierarchy.

![Figure 2. A 2-additive Choquet module with 2 inputs, involving three categories of hidden neurons noted A, B and C (see text). The weight of each edge is indicated. Figure taken from [2].](image)

2.4.2 Monotonic marginal utility module

A monotonic marginal utility module is a small multilayer perceptron, with a single hidden layer. It represents a monotonic marginal utility as a convex sum of logistic sigmoids, as found in [3]:

\[
u_i(x_i) = \frac{p}{1 + e^{-(r_i^\eta x_i - r_i^\beta)}}.
\]

(5)

Where \( p \) is a hyperparameter, which can be controlled through regularization. The parameters to be learned are the weights \( r \), the steepness parameters \( \eta \) and the biases \( \beta \). Note that, in order to validate the constraints given in Section 2.2, it is enough to ensure that the \( r_i^k \)'s be non-negative and sum to one, and that the \( \eta_i^k \)'s are all positive (resp all negative) for a non-decreasing (resp non-increasing) marginal utility.

3 Related work

A traditional way of learning marginal piecewise-linear utilities is the UTA methods [11], later generalized by GRIP [5, 4] in the context of ordinal regression, by allowing to learn any monotonic marginal utility. Another approach is the use of cubic splines, which offer both smoothness and representation power [10, 6]. All of these methods use linear programming and often benefit from, or require, an interaction with a field expert who can give selected preference examples as the methods iterate.
With no loss of generality, the domain $X$ is assumed to be $[-\infty, +\infty]$. The first extension we make is allowing for single-peaked (resp. previous) value of $M_i$. With a threshold composed of two monotonic utility modules, $M_1$ and $M_2$. 

Let $X_i \subseteq \mathbb{R}$ the input domain. A SPUM $u_i$ is a multilayer perceptron composed of two monotonic utility modules, $M_1$ and $M_2$, along with a threshold $t \in X_i$. $M_1$ (resp. $M_2$) is a non-decreasing module (resp. non-increasing). Then, for $x_i \in X_i$, we have:

$$u_i(x_i) = \begin{cases} M_1(x_i) & \text{if } x_i < t \\ M_2(x_i) & \text{otherwise} \end{cases}$$

The elements to learn are thus $t$, along with the parameters of $M_1$ and $M_2$. Note that learning a single-valleyed utility is exactly the same process; nonetheless the output node returns $1 - o$, with $o$ the output of a SPUM. This also means that the gradients are multiplied by $-1$ during backpropagation.

### 4.1 Bitonic utilities

With no loss of generality, the domain $X_i$ of criterion $x_i$ is assumed to be $[-\infty, +\infty]$. The first extension we make is allowing for single-peaked (resp. single-valleyed) marginal utilities to be learned. We call $u_i$ a single-peaked marginal utility if and only if:

1. $\lim_{x_i \to -\infty} u_i(x_i) = 0$
2. $\lim_{x_i \to +\infty} u_i(x_i) = 0$
3. $\exists t_i \in X_i$ s.t. $u_i$ is non-decreasing on $[-\infty, t_i]$ with; and $u_i$ is non-increasing on $[t_i, +\infty]$. Note that $t_i$ needs not be uniquely defined, i.e. $u_i$ might be constant around $t_i$.

Note that the normalization constraints of marginal utilities (see Section 2.2) impose that $u_i(t_i) = 1$. $u_i$ is called single-valleyed utility if $1 - u_i$ is single-peaked. We will describe below how we implement a single-peaked utility module (SPUM).

#### 4.1.1 Components of a SPUM

Let $X_i \subseteq \mathbb{R}$ the input domain. A SPUM $u_i$ is a multilayer perceptron composed of two monotonic utility modules, $M_1$ and $M_2$, along with a threshold $t \in X_i$. $M_1$ (resp. $M_2$) is a non-decreasing module (resp. non-increasing). Then, for $x_i \in X_i$, we have:

$$u_i(x_i) = \begin{cases} M_1(x_i) & \text{if } x_i < t \\ M_2(x_i) & \text{otherwise} \end{cases}$$

The stochastic update rule thus becomes:

$$t \leftarrow t - \rho_i \delta_i(t_i) \frac{\partial \mathcal{L}(x_i)}{\partial u_i(t_i)}$$

The backpropagation process.

### 4.1.3 Re-normalizations

After applying the gradients, the function needs to be re-normalized in order to fit the constraints of a single-peaked marginal utility. First of all, $M_1$ and $M_2$ are normalized as monotonic marginal utilities: the $\eta_k$ are clipped to the right sign, the negative $\eta_k$ are clipped to 0, and the other $\eta_k$ are normalized so that they sum to 1. The only remaining constraints to be filled is then that $u_i(t_i) = 1$, and that $u_i$ be continuous. As $M_1$ and $M_2$ are respectively continuous, it is enough to have $M_1(t) = M_2(t) = 1$, which is easily achieved by dividing the weights $r^k_1$ of $M_1$ (resp $M_2$) by $M_1(t)$ (resp $M_2(t)$).

To avoid such a process of interaction, and when data is available, statistical methods can be used. A probabilistic approach for learning an additive utility model (a special case of the Choquet Integral) can be found in [1]. [3] uses a quadratic-programming solver to learn a single 2-additive Choquet integral along with monotonic marginal utilities, which [2] later generalized by allowing to learn Hierarchical 2-additive Choquet integrals.

#### 4.1.2 Backpropagation

At each parameters update, both $M_1$ and $M_2$ are updated normally, with the same gradient as if they were any monotonic utility. As $u_i$ is not necessarily differentiable in $t$, we cannot compute a real gradient for $t$. We instead use a surrogate value $\delta_i(t_i)$ for the gradient, computed as :

$$\delta_i(t_i) = (x_i - t)$$

4.1.2 Backpropagation

At each parameters update, both $M_1$ and $M_2$ are updated normally, with the same gradient as if they were any monotonic utility. As $u_i$ is not necessarily differentiable in $t$, we cannot compute a real gradient for $t$. We instead use a surrogate value $\delta_i(t_i)$ for the gradient, computed as :

$$\delta_i(t_i) = (x_i - t)$$

The stochastic update rule thus becomes:

$$t \leftarrow t - \rho_i \delta_i(t_i) \frac{\partial \mathcal{L}(x_i)}{\partial u_i(t_i)}$$

with $\rho_i$ the learning rate for $t$ and $\mathcal{L}$ the gradient immediate loss function.

#### 4.1.3 Re-normalizations

After applying the gradients, the function needs to be re-normalized in order to fit the constraints of a single-peaked marginal utility. First of all, $M_1$ and $M_2$ are normalized as monotonic marginal utilities: the $\eta_k$ are clipped to the right sign, the negative $\eta_k$ are clipped to 0, and the other $\eta_k$ are normalized so that they sum to 1. The only remaining constraints to be filled is then that $u_i(t_i) = 1$, and that $u_i$ be continuous. As $M_1$ and $M_2$ are respectively continuous, it is enough to have $M_1(t) = M_2(t) = 1$, which is easily achieved by dividing the weights $r^k_1$ of $M_1$ (resp $M_2$) by $M_1(t)$ (resp $M_2(t)$).

To avoid such a process of interaction, and when data is available, statistical methods can be used. A probabilistic approach for learning an additive utility model (a special case of the Choquet Integral) can be found in [1]. [3] uses a quadratic-programming solver to learn a single 2-additive Choquet integral along with monotonic marginal utilities, which [2] later generalized by allowing to learn Hierarchical 2-additive Choquet integrals.
At each backpropagation step, the parameters of $M$ are not in its optimal state at that point. Another criterion would be re-characterizing by $M_2$ (resp. $M_1$) if $t$ is $\delta$-close to 0 (resp. 1) for $k$ iterations. This, nonetheless, is very sensitive to small/large values of the "real" threshold, and we found it to be much less stable than the one above.

4.2 Selectors

Our second contribution is the introduction of a module that allows to automatically select the best-fitted utility type, among single-peaked, single-valleyed, non-decreasing or non-increasing. We call such modules selectors.

4.2.1 Components of a selector

Let $X_i \subseteq \mathbb{R}$ the input domain. A selector $u_i$ is a multilayer perceptron composed of two utility modules, $M_1$ and $M_2$, along with a switch $s \in X_i$. $M_1$ (resp $M_2$). Supposedly, $M_1$ and $M_2$ have different types. $u_i$ will also have another parameter, called the switch $s \in \mathbb{R}$. Then, for $x_i \in X_i$, we have:

$$u_i(x_i) = l(s) M_1(x_i) + (1 - l(s)) M_2(x_i)$$

Where $l : x \rightarrow \frac{1}{1 + e^{-x}}$ is the logistic sigmoid, ensuring $l(s) \in [0, 1]$. $u_i$ is thus a convex sum of both possible models. $l(s) = 1$ (resp 0) means that the model favors $M_1$ (resp $M_2$).

4.2.2 Backpropagation

At each backpropagation step, the parameters of $M_1$ and $M_2$ are updated normally, with the usual gradients simply weighed by $l(s)$ (resp $(1 - l(s)$)), $s$ is thus updated as:

$$s \leftarrow s - \rho_s \cdot \frac{\partial l(s)}{\partial s} \cdot \frac{\partial u_i(x_i)}{\partial l(s)} \cdot \frac{\partial L(x_i)}{\partial u_i(x_i)}$$

$$= s - \rho_s \cdot l(s)(1 - l(s)) \cdot (M_1(x_i) - M_2(x_i)) \cdot \frac{\partial L(x_i)}{\partial u_i(x_i)}$$

with $\rho_s$ the associated learning rate.

4.2.3 Re-characterization

It is obvious that, as long as $l(s)$ is neither 0 or 1, $u_i$ has no reason to be a valid marginal utility. As a consequence, it is necessary to establish a criterion to select either marginal utility.

We apply the following strategy: we choose a margin $\epsilon$ and a stability parameter $k$. Then, if $l(s)$ remains $\epsilon$-close to 0 (resp 1) for $k$ epochs, $M_2$ (resp $M_1$) is chosen, and takes the place of $u_i$ for the subsequent iterations.

The main difference with the re-characterization of single-peaked utilities is that, here, selecting either model is necessary in order to have a valid model. As a consequence, the selection must be forced at some point during the training, for instance, when a certain number of epochs is reached.

In any case, it is preferable to keep training for some times after such a re-characterization (or to re-train completely using the new marginal utility), as there are chances that the selected module will not be in its optimal state at that point.

4.2.4 Discussion

In practice, one can use any two marginal utility modules for $M_1$ and $M_2$. The two most usual cases, nonetheless, would be:

- (1): $M_1$ non-decreasing and $M_2$ non-increasing, when $u_i$ is known to be monotonic
- (2): $M_1$ single-peaked and $M_2$ single-valleyed

Note that case (2) is the most general: through the process defined in Section 4.1.4, this module can also learn a monotonic utility.

5 Experimental Setting

5.1 Selectors

The tests below were made on artificial data, to test the ability of the new modules to learn models which are known perfectly. In order to train the selectors, we create a UHCI model $G$, whose parameters we know. Note that the model is not a Neur-HCI network, as the marginal utilities are explicitly defined as mathematical functions; this means that they can only be approached by a Neur-HCI with a finite number of sigmoids (we use 100 in these experiments).

We generate random alternative by drawing random vectors in the unit hypercube of the right dimension, and compute the label of each alternative as the score given by $G$ on this alternative.

Finally, we train 100 Neur-HCI models with the most general selectors in place of utility modules ($M_1$ is single-peaked, $M_2$ is single-valleyed). Note that a new dataset is drawn for each model. These selectors can choose between the four types of marginal utilities, both monotonic and bitonic ones. We then compare the type of the learned utilities to their ground-truth type. The results are presented in the confusion matrices below. Each line and column correspond to a type of marginal utilities, with: "ND": non-decreasing; "NI": non-increasing; "SP": single-peaked and "SV": single-valleyed. For instance, the number in square from row "ND" and column "SV" shows how many times the model learned a single-valley function when the ground truth was "non-decreasing".

<table>
<thead>
<tr>
<th></th>
<th>ND</th>
<th>NI</th>
<th>SP</th>
<th>SV</th>
</tr>
</thead>
<tbody>
<tr>
<td>ND</td>
<td>0.993</td>
<td>0.0</td>
<td>0.007</td>
<td>0.0</td>
</tr>
<tr>
<td>NI</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>SP</td>
<td>0.0</td>
<td>0.01</td>
<td>0.99</td>
<td>0.0</td>
</tr>
<tr>
<td>SV</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table 1. Confusion matrix, 6 criteria, 1000 generated examples

We see that there are very few errors here, showing Neur-HCI ‘s ability to learn the right type of utilities. This is coherent with the identifiability of a UHCI fitted with utilities of these types [9]. Moreover, there is never confusion between “opposite” types of utilities: a non-increasing one (resp. single-peaked) is never identified as a non-decreasing (resp. single-valleyed) one, and vice versa.

5.2 Bitonic utilities

We trained 100 models (4-dimensional, with one marginal utility of each type) on 1000 randomly generated examples (different for each models). Figure 4 shows the graph of all of the learned marginal utilities (100 per graph), with the ground-truth values as stars. As we see, there is little variation between the learned functions, which fit the ground-truth well. The most sensitive areas seem to be the threshold in the bitonic case, as the maximal variance is observed there.
Note that the parameters of the aggregators also showed very little variance and fitted the ground truth model well (the maximal error between a learned weight and its true value was 0.02, the average error on all weights and all models was 0.013).

5.3 Discussion

We saw above that NEUR-HCI is able to recognize the direction of marginal utilities (among the four defined behaviours) in most cases, with artificial data. Moreover, it yields high stability both in the parameters of the aggregators and the values of the learned marginal utility functions. This opens an opportunity in using NEUR-HCI as an analysis tool; in a case where a user would wish to know whether the type of the relationship between several criteria and an output (or a label), a NEUR-HCI can be trained on a dataset and show which values increase (or decrease) the output. In domains where the user can affect the value of the criteria for building alternatives (in a design setting, for instance), this can in turn be used to optimize the values of the criteria w.r.t. a given metric; by increasing (resp. decreasing) the criteria with an increasing (resp. decreasing) utility, or by aiming towards the peak (resp. valley) when the utility is bitonic. By doing this, a user could build better alternatives or improve existing ones.

This could also find application in black box optimization model, where parameters can be meaningless (and thus not have an obvious relationship with the output), in order to find whether several parameters tend to affect the output positively or not, and thus indicating directions or regions to explore, using the parameters of the aggregator as indicator of their relative importance.

6 Conclusion

This paper proposes an extension of the NEUR-HCI framework which allows to, firstly, learn bitonic marginal utilities, and secondly automatically select the best type of marginal utility for a given criterion. Experiments run on artificial data show accuracy in the selection of utilities and high precision in their values, fitting well the ground truth values. This is a first step in validation, but tests on real data are a logical follow-up to study noise robustness. Stability tests with noisy examples, with varying data volumes and dimensions could prove relevant to evaluate the limits of the model.

These preliminary results nonetheless encourages the generalization of NEUR-HCI modules to other models, such as new aggregators built with the same techniques, and always ensuring the fulfillment of all constraints.

REFERENCES